RL-TR-93-200
Final Technical Report
October 1993

PARALLEL ASSESSMENT WINDOW SYSTEM (PAWS) ENHANCEMENTS

Syracuse University

Daniel J. Pease

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: Milissa M. Benincasa

MILISSA M. BENINCASA
Project Engineer

FOR THE COMMANDER

JOHN A. GRANIERO
Chief Scientist for C3

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1993	Final

4. TITLE AND SUBTITLE
PARALLEL ASSESSMENT WINDOW SYSTEM (PAWS) ENHANCEMENTS

5. FUNDING NUMBERS
C - F30602-91-D-0001
PE - 62702F
PR - 5581
TA - 18
WU - P4

6. AUTHOR(S)
Daniel J. Pease

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Syracuse University
Center for Science and Technology
Syracuse NY 13244

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-93-200

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Milissa M. Benincasa/C3CB/(315) 330-7650

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public r[illegible]; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This final technical report documents work conducted under the Expert Engineering and Science Program (ES&E) entitled "Parallel Architecture Assessment Tool". This report contains a summary of the activities conducted under this effort which included the enhancement of the Parallel Assessment Window System (PAWS). PAWS is an experimental system for performing machine evaluation and comparisons of parallel architecture computers against programmed algorithms. This report provides a comprehensive users manual for the PAWS tool.

14. SUBJECT TERMS
High Performance Computing, Parallel Performance Assessment, Dataflow Graphs, Parallel Architectures, Visualization Techniques

15. NUMBER OF PAGES
52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Final Report

for PAWS Enhancement Contract through SRI

Executive Summary

This report is the Final Report contract deliverable for the "PAWS Enhancement Contract" from SRI for Rome Laboratories.

A modified version of the Parallel Assessment Windowing System (PAWS) that contains all of the enhancements developed throughout this contract has been delivered to Rome Labs.

This document contains a summary of the work completed, the final status of tasks defined for the project, an overview of the enhanced version of PAWS, and the complete Users Manual for Version 1.0 of PAWS.

Summary of SU Activities

Dr. Daniel Pease, Associate Professor of Electrical and Computer Engineering at Syracuse University, was the Responsible Individual for the contract. He was responsible for the technical content and technical management of the work performed.

Mr. Mikki (Visualization and Parallel Mapping), Mr. Foudil-Bey (Math Modeling and Assessment), and Mr. Zerrouki (Language to IF1 Conversion and Architectural Characterization) refined PAWS and enhanced its capabilities.

Mr. Mikki refined the visualization tools to include SIMD and distributed architectures, manual partitioning, and added automated linking to Architecture Data Base for communication weights of partitioned elements.

Mr. Zerrouki upgraded the Ada to IF1 Converter, integrated his new refinements with the other parts of PAWS, and tested their impact and correctness.

Mr. Foudil-Bey refined the Assessment System to deal with asynchronous operations for message passing and to integrate it with the other sections of PAWS.

Status of Short-term and Long-Term Technical Tasks

Architectural Characterization

1. Migrate the Generic Parallel System Data Base to the Sun4/Sparc Station environment.
   Schedule: Start by: February 15, 1991
   Completed: March 15, 1991
   Status: Completed: March 15, 1991

2. Refine the Architectural Tools to work in Open View Environment.
   Schedule: Start by: May 1, 1991
   Complete by: June 1, 1991
   Status: Completed: June 15, 1991

3. Characterize a Hyper Cube Message Passing Architecture
   Schedule: Start by: March 15, 1991
   Complete by: September 15, 1991
   Status: Completed: September 15, 1991

Language to IF1 Conversion

1. Migrate the Ada to IF1 Converter to the Sun4/Sparc Station environment.
   Schedule: Start by: February 15, 1991
   Completed: March 15, 1991
   Status: Completed: March 15, 1991

2. Refine the IF1 interpreter to work in Open View Environment.
   Schedule: Start by: May 1, 1991
   Complete by: June 1, 1991
   Status: Completed: June 1, 1991

3. Refine the Design of an ADA to IF1 interpreter.
   Schedule: Start by: June 1, 1991
   Complete by: December 1, 1991
   Status: Completed: December 1, 1991

4. Refine the Design of an Lisp to IF1 interpreter.
   Schedule: Start by: July 1, 1991
   Complete by: September 1, 1991
   Status: Refinement Complete October 1, 1991

Status of Short-term and Long-Term Technical Tasks(Continued)

Visualization and Parallel Mapping

1. Migrate the Visualization Tools to the Sun4/Sparc Station environment.
   Schedule: Start by: February 15, 1991
   Completed: March 15, 1991
   Status: Completed: March 15, 1991

2. Refine the Visualization Tools to work in Open View Environment.
   Schedule: Start by: May 1, 1991
   Complete by: June 1, 1991
   Status: Completed: June 1, 1991

3. Refine the Visualization Tools to Provide more types of Partitioning
   Schedule: Start by: March 15, 1991
   Complete by: September 15, 1991
   Status: Completed: September 15, 1991

4. Include Visualization of Execution with Manual Partitioning
   Schedule: Start by: September 15, 1991
   Complete by: February 1, 1992
   Status: Completed: February 1, 1992

Math Modeling and Assessment

1. Migrate the Math Modeling and Assessment Tools to the Sun4/Sparc Station environment.
   Schedule: Start by: February 15, 1991
   Completed: March 15, 1991
   Status: Completed: May 15, 1991

2. Refine the Assessment Tools to work in Open View Environment.
   Schedule: Start by: May 1, 1991
   Complete by: July 1, 1991
   Status: Completed: June 15, 1991

3. Refine the Assessment Tools to Provide better Significance Measures
   Schedule: Start by: March 15, 1991
   Complete by: September 15, 1991
   Status: Completed: September 15, 1991

4. Refine the Math Model to deal with Asynchronous Operations for Message Passing
   Schedule: Start by: September 15, 1991
   Complete by: February 1, 1992
   Status: Completed: February 1, 1992

Overview of Use and Operation of Version 1.0 of the Parallel Assessment Windowing System (PAWS)

Introduction

Version 1.0 of Parallel Assessment Windowing System (PAWS) is a refined version of prototypes originally developed for Rome Laboratories (RL) and DARPA. These refinements were funded by RL. The release of Version 1.0 means that PAWS is now an integrated tool that can be used to assist software developers and system acquisition personnel to assess parallel systems.

PAWS runs on Sun SparcStations under UNIX and uses Open Windows. It is easily installed and provides an interactive, visual environment for users who develop Ada programs to assess their parallelism. PAWS provides users three fundamental capabilities:

1. Select the best parallel system for their existing Ada applications.

2. Evaluate and visualize the parallelism that exists in their existing Ada applications.

3. Assist users in learning about parallelism and in manipulating their programs to execute more effectively on parallel systems.

In this overview each of these will be discussed in detail. However, first the basic operating sequence of PAWS will be introduced.

Operating Sequence

The basic operation of PAWS is shown in Figure 1.

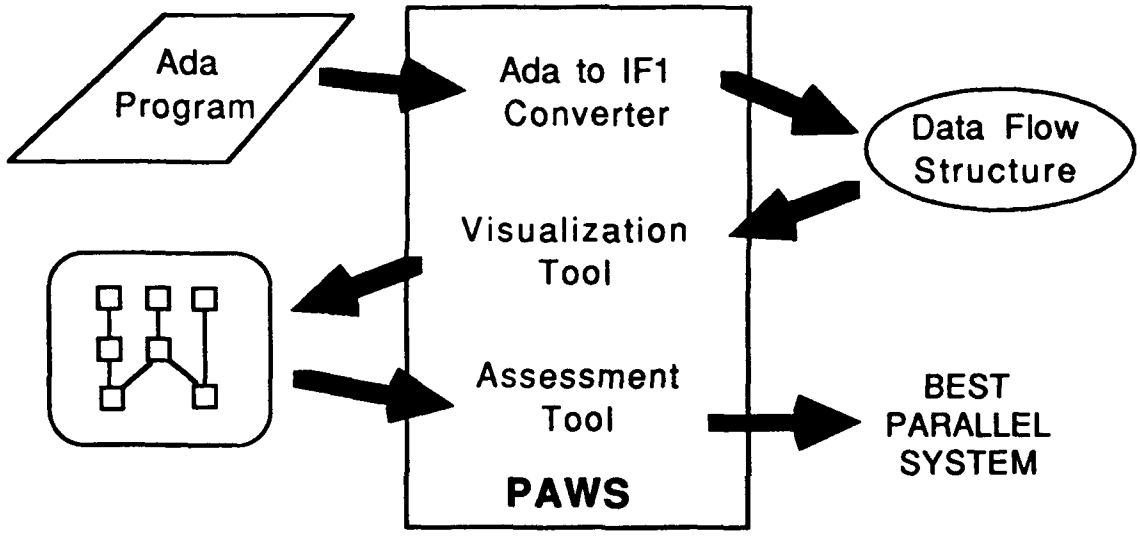


Figure 1 - Basic Operation of PAWS

There are three sections to PAWS, which are usually used in order. These sections are:

1. Ada-to-IF1 Converter
2. Visualization Tool
3. Assessment Tool

Starting with an executable Ada program the user converts the program into a data flow structure with the Ada-to-IF1 Converter. The data flow structure is a representation of the program that is machine independent and can be analyzed to detect the parallelism expressed in the program. Every user of PAWS starts by converting their programs into a data flow structure.

The user can then look at the parallelism in his program by visually following its parallel execution with the Visualization Tool. The Visualization Tool provides a high-level view of parallelism for the user. They can get a feel for its parallel execution. In addition they can manipulate their program to be executed in different ways on different parallel systems.

Finally the user can look at the parallel execution of their program in technical detail. This is done with the Assessment Tool. This tool provides detailed profiles and analysis of the expected execution of the users program on different parallel systems. These results are then analyzed by the Assessment Tool to identify the best parallel system.

Select the Best Parallel System

This perspective is aimed at the user that just wants to make a detailed assessment to identify which specific Parallel system is the best for their application. Figure 2 shows the path of usage for this pure assessment.

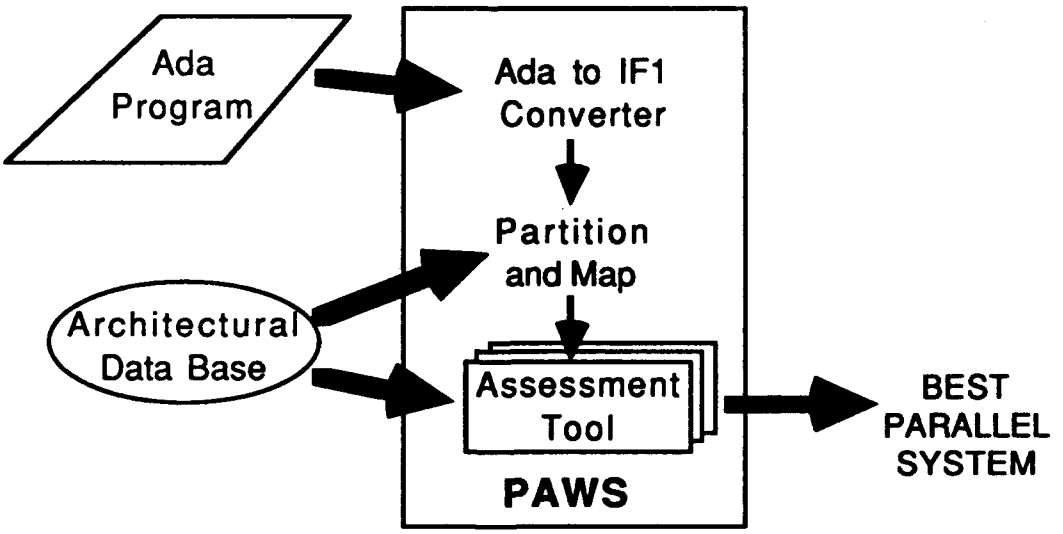


Figure 2 - Using PAWS to Select Best Parallel System

The user takes their Ada program and converts it into a data flow graph with the Ada to IF1 Converter. This can be done in pieces, but has to be done in accordance to the hierarchy of the program, starting at the top.

Then the Visualization Tool is used to partition the graph into pieces that can execute independently and to map the graph to several architectures. The Visualization Tool will do this partitioning and mapping automatically based on built-in optimization algorithms. There will be a separate mapping for each different architecture. The partitioning and mapping results are saved in data files which are then passed to the Assessment Tool.

The Assessment Tool will then perform a detailed analysis of the execution performance of the data flow graph on the different architectures using the mapping and partitioning information and architectural parameters from the architectural data base. The results of this analysis will then be compared by the Assessment Tool and the most suitable parallel system is identified. This is an automatic process with a ranking produced of all the architectures considered.

Evaluate and Visualize Parallelism

This perspective is aimed at the user that just wants to make a detailed analysis of the performance of a particular architecture for their application. The user will be generating a great deal of information that is presented to them for their evaluation. It is assumed that the user will understand what performance data is and its interpretation is left to the user. Figure 3 shows the path of usage for this detailed performance assessment.

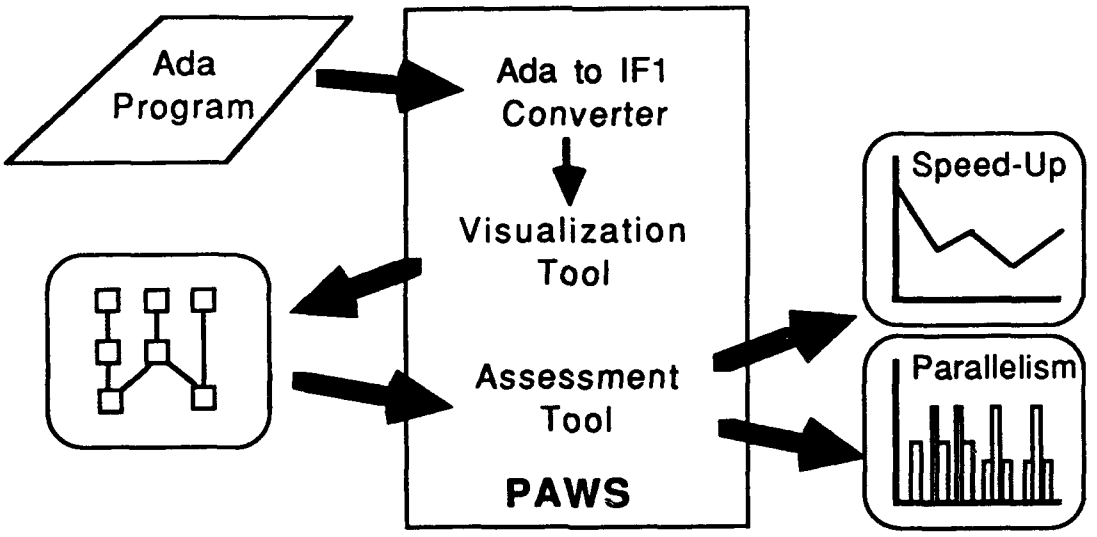


Figure 3 - Using PAWS to Assess Detailed Performance of Program

The user takes their Ada program and converts it into a data flow graph with the Ada to IF1 Converter. This can be done in pieces, but has to be done in accordance to the hierarchy of the program, starting at the top.

Then the Visualization Tool is used to partition the graph into pieces that can execute independently and to map the graph to several architectures. The Visualization Tool will do this partitioning and mapping automatically based on built-in optimization algorithms. The user will be able to intervene at this point and manually partition their programs if they want to evaluate different possible partitions and their resultant mappings. Each partition will be a separate mapping for each different architecture. The partitioning and mapping results are saved in data files which are then passed to the Assessment Tool.

The Assessment Tool will then perform a detailed analysis of the execution performance of the data flow graph on the different architectures using the mapping and partitioning information and architectural parameters from the architectural data base. The results of this analysis will then be displayed to the user. Different iterations of this analysis with variations in parameters like the sizes of loop indices can be performed. All of the results can be saved. Results from multiple analysis can be displayed simultaneously in different windows so they can be compared. Most of the results are presented in graphical form.

Assist Users in Learning about Parallelism

This perspective is aimed at the user that wants to learn how to write more effective parallel programs. Figure 3 shows the path of usage for this detailed performance assessment.

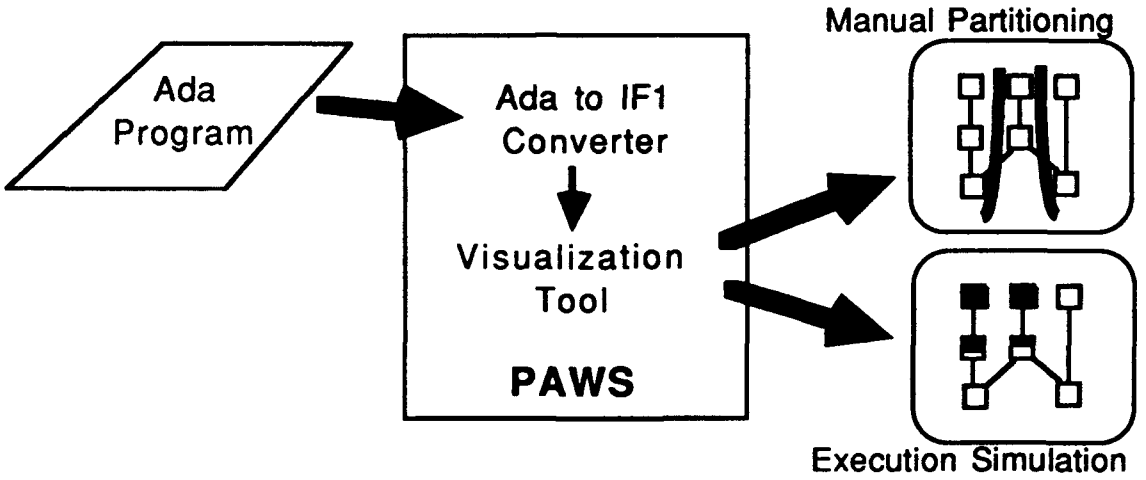


Figure 4 - Using PAWS to Learn About Parallelism

The user takes their Ada program and converts it into a data flow graph with the Ada to IF1 Converter. This can be done in pieces, but has to be done in accordance to the hierarchy of the program, starting at the top.

Then the Visualization Tool is used to identify the parallelism in the data flow graph representation of their program. This parallelism will be displayed to the user as a graphic representation of the data flow in their program. The user can then simulate the execution of their program on several generic architectures in order to see how well it executes and what the problem areas are that restrict its execution on a particular class of architecture. The classes that can be simulated are:

Sequential Uniprocessor,
Single Instruction Multiple Data Stream Parallel Processor (SIMD),
Multiple Instruction Multiple Data Stream Parallel Processor (MIMD) with task synchronization,
Multiple Instruction Multiple Data Stream Parallel Processor (MIMD) with synchronization at the node level, and
Multiple Instruction Multiple Data Stream Parallel Processor (MIMD) with manual partitioning.

The Visualization Tool will show the user synchronization penalty and the relative performance of the different parts of their program so they can learn to partition in a manner that leads to load balancing. These capabilities can be reused over and over with different versions of their program. Multiple windows with the versions can be managed on the screen so the different versions can be compared.

Hybrid Use of PAWS

Features of each of the three perspectives of PAWS can be used at the users discretion. As the user learns more about parallelism they can perform more detailed analysis of their programs using the Assessment Tool. All of the capabilities of PAWS can be used at any time, as long as the fundamental chain of order is maintained. This chain is:

Step 1 - No visualization, partitioning, mapping, parallelism detection or assessment can be done until the user's Ada program has been run through the Ada to IF1 Converter.

Step 2 - No assessment can be done until the data flow graph of the user's Ada program has been partitioned and mapped by the Visualization Tool.

Step 3 - No determination of best architecture can be made until the Assessment Tool is used.

PAWS USER'S MANUAL

A Software Development Tool For Assessment of Parallel Computers and Parallel Programs

Release 1. 1
October 29th, 1992

Developed by

Prof Dan Pease
Kamal Foudil-Bey
Mohammed Mikki
Mohamed Zerrouki

at

Syracuse University

for

Rome Laboratory

1 PREREQUESITES FOR THE INSTALLATION OF PAWS:

PAWS requires the following items to run:

- The X Window system in an openwindow environment
- YACC: A language parser generator.
- LEX: a language lexical generator.

2 INSTALLING PAWS

PAWS resides in a global directory called PAWS which in turn contains all the subtools that make up the PAWS package. Its installment requires the installment of all the directories pertaining to each tool. The installment of each subtool is discussed in the following sections.

Figure 1 shows the directory structure of the PAWS software. For a successful execution the following environment variable must be defined in the .cshrc file:

setenv MYPAWS $HOME/<OPTIONAL PATH>/PAWS

The OPTIONAL PATH refers to any path between the user home directory and the PAWS directory. For example if PAWS resides in the directory user_directory/ALPHA/BETA/PAWS then the optional path would be ALPHA/BETA
The following directories contain source code with their associated Makefile:

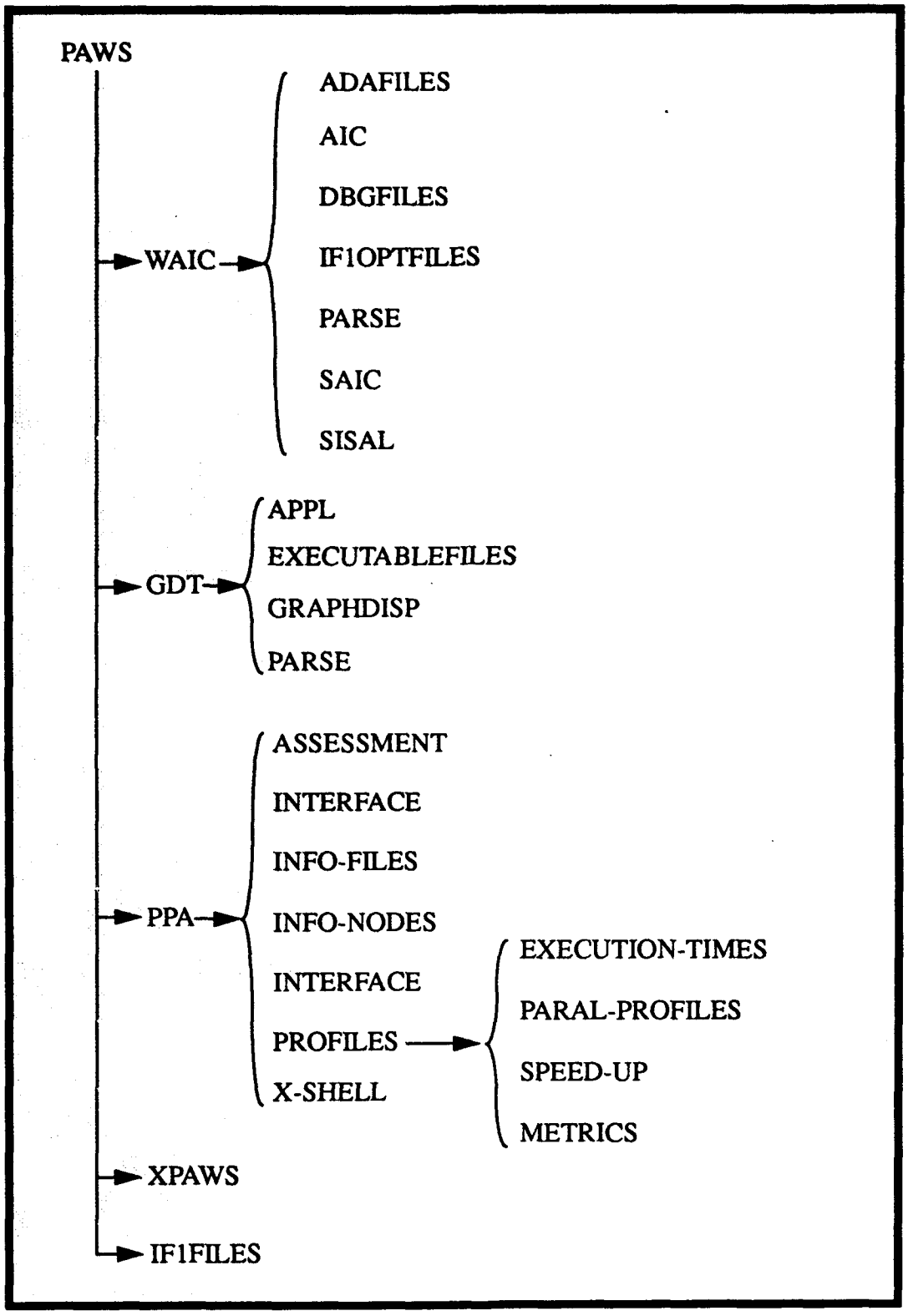
- $MYPAWS/GDT/APPL
- $MYPAWS/GDT/PARSE
- $MYPAWS/GDT/GRAPHDISP
- $MYPAWS/GDT
- $MYPAWS/PPA/INTERFACE
- $MYPAWS/PPA/X-SHELL
- $MYPAWS/PPA/ASSESSMENT
- $MYPAWS/WAIC/AIC
- $MYPAWS/WAIC/SAIC
- $MYPAWS/XPAWS

If any changes are made to the source code in any of the directories cited above, the user must recompile the appropriate section by invoking the" make" command

2.1 Description of Directory Content

$MYPAWS/WAIC/ADAFILES: This directory contains Ada files which are used to convert Ada code to IF1 code. Any Ada application must reside in this directory in order to be accessible by the converter.

$MYPAWS/WAIC/AIC: This directory contains the programs necessary for the



conversion of Ada code to IF1 code. There are five categories of files:

FIGURE 1 Paws Directory Structure Organization

- Files storing the lexical and syntactic specifications of the Ada language.
- Header files for all the common variables used by the converter programs.
- Files storing the semantical routines for the converter. These files have the prefix "IF1_".
- Files generated when the tools LEX and YACC are applied to the specification files.
- File containing a routine for syntax error detection in Ada programs.

$MYPAWS/WAIC/DGBFILES: This directory contains three categories of debugging files generated by the converter:

- Files that store the symbol table of the Ada to IF1 converter, which displays all the variables used in the Ada source code. It is a block structured symbol table, where each block corresponds to a specific scope of the Ada code. These files have the "tbl" extension.
- Files that store the scope tree of the Ada source code according to the constructs used. Each compound construct in Ada has a scope associated with it. This tree illustrates the relationship between the scopes in a parent-child fashion. These files have the "scp" extension.
- Files that store the trace of the conversion of Ada constructs to IF1 primitives. It provides a history of all actions taken by the converter. These files have the "trc" extension.

$MYPAWS/WAIC/IF1OPTFILES: This directory contains files obtained by applying the OSC (Optimizing Sisal Compiler) tool provided by Lawrence Livermore National Laboratory. There are six categories of files:

- Files that store opimized IF1 code. These have the "opt" extension..
- Files that store a non-dataflow version of IF1 called IF2. Theses have as suffix "mem".
- Files that store another intermediate form similar to IF1 but has more non-dataflow nodes to it (memory management nodes). These have the "up" extension.
- Files that store IF2 code with partitioning data associated with all the components of IF1 graphs. These have the "part" extension.
- Files that store an equivalent C code of the IF1 code. These have the "c" extension.
- Files that store the equivalent Assembly code. These have the "s" extension.

$MYPAWS/WAIC/PARSE: This directory contains the programs needed for a complete Ada parser. These programs were built using Ada specification files obtained from other sources. These programs do not perform any conversion but check for syntax errors in Ada programs.

$MYPAWS/WAIC/SAIC: This directory contains the programs necessary to gener-

ate the window environment which allows the users to run the application characterization tool. The window environment is a set of menus designed for for selecting different options that the tool supports.

$MYPAWS/WAIC/SISAL: This directory contains Sisal programs that are supplied by the users. Sisal is a data flow experimental language. It is a highly parallel language developed by Lawrence Livermore National Laboratory. IF1 is the program graph of the Sisal language.

$MYPAWS/GDT: In addition to containing the above three directories this directory contains the following items.

- The executable file generated from using the "make" command in the $MYPAWS/GDT/APPL directory.
- The executable file generated from using the "make" command in the $MYPAWS/GDT/PARSE directory.
- The executable file generated from using the "make" command in the $MYPAWS/GDT/GRAPHDISP directory.

$MYPAWS/GDT/APPL: This directory contains the following items.

- The main menu for the GDT.
- The main menu for selecting the available IF1files.

$MYPAWS/GDT/EXECUTABLEFILES

$MYPAWS/GDT/GRAPHDISP: This directory contains the following items.

- IF1 Graphical display tool.
- All programs dealing with partitioning, graphical display, execution simulation etc.

$MYPAWS/GDT/PARSE: This directory contains the following item.

- Data conversion from IF1 to C-Structure for visual display purposes.

$MYPAWS/PPA/ASSESSMENT: This directory contains the following items:

- Assessment program: This program is the core of the assessment tool. It performs a graph walk algorithm and generates several data files for display and

analysis. All the output files generated by the assessment code are stored in the directory $MYPAWS/PPA/PROFILES

$MYPAWS/PPA/DATA-FILES: This directory contains the following items.

- Data files shown in figure 2

$MYPAWS/PPA/INFO-NODES: This directory contains the following items:

- Files that have the following format: "<main | procedure name>.<int>". As an example "main.26" will contain information relevant to compound node or CALL node number 26 in the main application.
    - If node number 26 is a compound node, then the file main.26 will have the following fields:
      <node name>,<iteration number>, <status>.
      Node name will be either LOOPA, LOOPB, FORALL, SELECT, or TAGCASE.
      Iteration number will be the upper bound of the number of iterations for LOOPA and LOOPB type nodes.
      Status is a boolean variable that specifies if a loop slicing strategy must be adopted for that particular compound node. Possible values are TRUE or FALSE.
    - If node number 26 is a CALL node (i.e, procedure call) then the file main.26 will have the following fields:
      <node name>,<iteration number,<status>

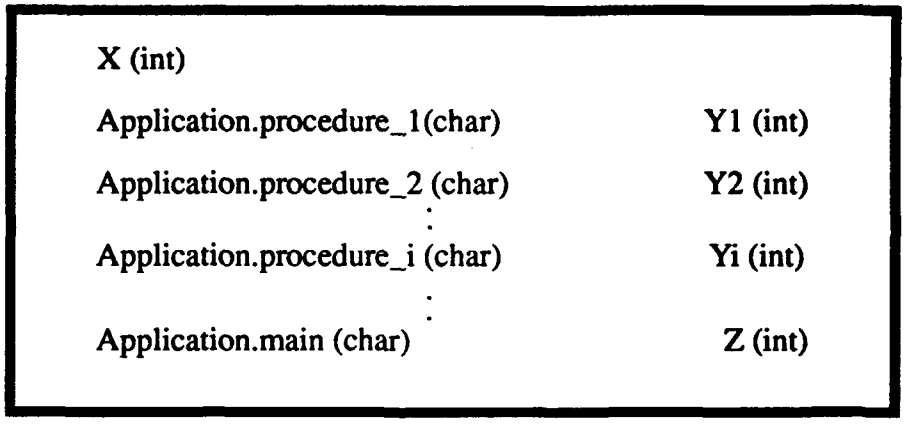


FIGURE 2 Filename data Format

$MYPAWS/PPA/INFO-FILES: This directory contains ASCII type format files where information relevant to each application is stored. (See section 6.1.6)

$MYPAWS/PPA/INTERFACE: This directory contains the following items:

- The program generating input data for the assessment program. The data is obtained partially from the output of the program "parse"(available as a subprogram in the GDT tool). The Interface code also performs automatic generation of several data files that are used by all programs residing in the

directory PPA. The interface code uses as input an IF1 file(Application.if1) from the directory
$MYPAWS/PAWS/IF1FILES and generates the following data files:

- Application.data
- Application.main

Figure 2 shows a typical Application.data format. The total number of functions (including the main function) is represented by X. Yi is the number of nodes included in procedure i. Z is the number of nodes included in the main function. The files listed in Application.data will reside in the directory $MYPAWS/PPA/DATA-FILES. These files contain ASCII code corresponding to the IF1 graph of the ADA program being analyzed.

$MYPAWS/PPA/PROFILES/EXECUTION-TIMES: This directory contains the following items:

- All main function and procedure execution times.

$MYPAWS/PPA/PROFILES/METRICS: This directory contains report files in which a summary of metrics are collected. This function is not fully implemented in this version.

$MYPAWS/PPA/PROFILES/PARAL-PROFILES: This directory contains output files generated by the Assessment program for displaying parallel profiles. All files in this directory will have the prefix "profile_". These files are used by the xgraph program to plot the parallelism profile of an application or any one of its compound nodes or procedure.

$MYPAWS/PPA/PROFILES/SPEED-UP: This directory contains output files generated by the Assessment program for displaying speed_up profiles. All files in this directory will have the prefix "speed_up_". These files are used by the xgraph program to plot the speed_up curve of an application or any one of its compound nodes or procedure.

$MYPAWS/XPAWS: This directory contains the source code for a user friendly interface initializing PAWS in an X Windows environment.

$MYPAWS/IF1FILES: This Directory contains IF1 files generated by the Ada converter.

$MYPAWS/PPA/X-SHELL: This directory contains the following items:

- Xassess program: This program is the user friendly interface developed under X_Windows to facilitate the input/output of data relevant to the assessment program. It consists mainly of popup windows with menus and submenus.

3 Installing and Starting PAWS

Once the directories have been created, the "make" command must be invoked in each directory where source code exists. If any change is made to the source code, then the code in that directory must be recompiled.

3.1 Preliminaries

PAWS can be invoked in 2 different ways. A user can execute PAWS by typing the "$MYPAWS/XPAWS/Welcome"
If PAWS is to reside in a specific location for an extended period of time, then it is best to start PAWS from the OpenLook window manager. This can easily be accomplished by adding the following lines to the file openwin-menu used by your system, for more information see your system administrator:
"START PAWS ..." exec $MYPAWS/XPAWS/Welcome
Once PAWS has been started the "welcome window" is displayed offering the user the choice to start PAWS or exit (Please figure 3).

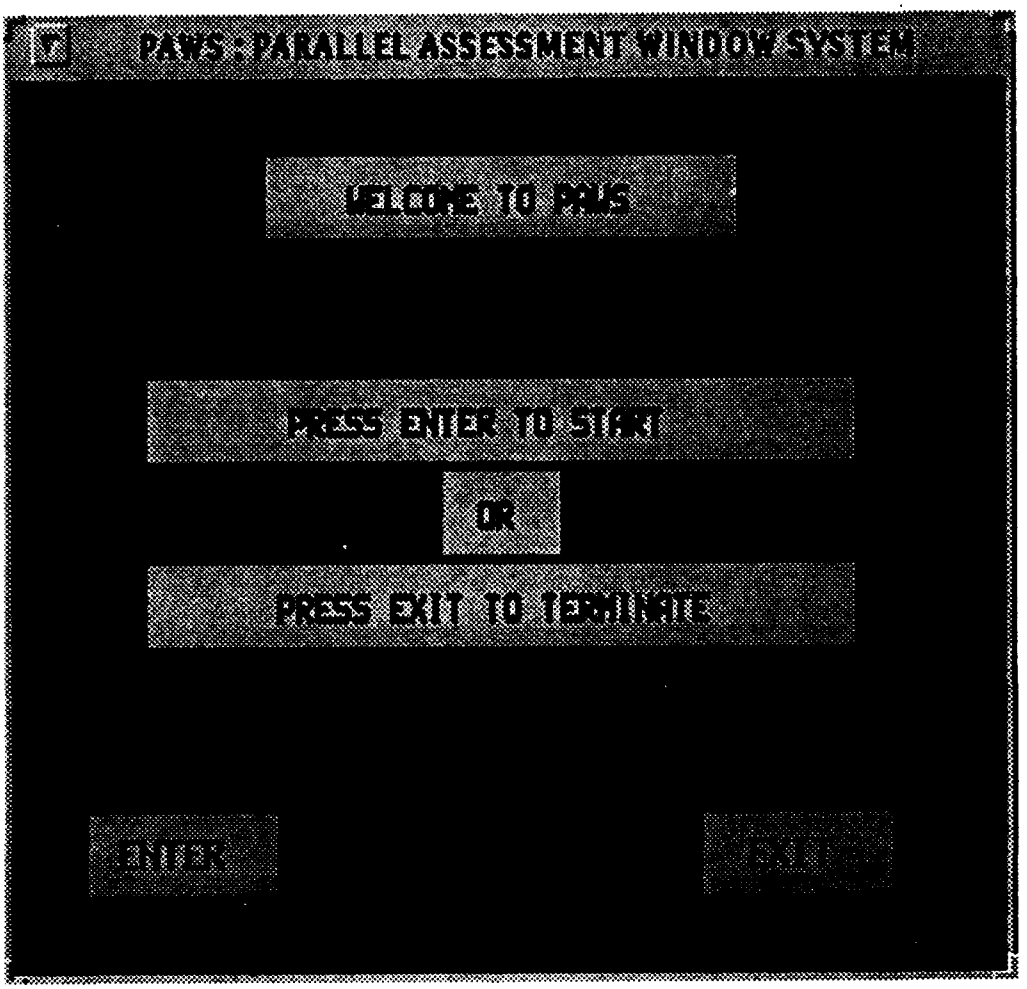


FIGURE 3 Welcome Window

By moving the mouse pointer over "ENTER", (in figure 3) , and clicking the left mouse button, the tool menu window is displayed (see figure 4).The user can execute any of the tools from the list.

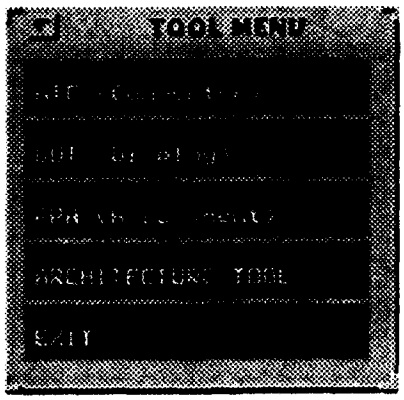


FIGURE 4 Tool Menu Wnndow

4 Running the Application Characterization Tool (AIC)

The application characterization tool is the cornerstone of PAWS. It takes an application and converts it into a parallel dataflow IF1 graph. The application charcaterization tool can be started by moving the mouse pointer over "AIC (Converter)" in the tool menu and clicking the left mouse button. The AIC main menu will be displayed, (see figure 5).
Applications written in ADA or SISAL can be transformed into IF1 code by selecting one of the first two menu options,"ADA to IF1 CONVERSION' or "SISAL to optimize IF1 CONVERSION". The other options "IF1 CODE OPTIMIZATION" is used to code on a SUN machine, and "IF1 CODE EXECUTION ON SUN" is used to run code on a SUN machine.
A message window shown in light gray reports to the users the selections from the different menus, as well as the start and end of every selection performed in the menu.

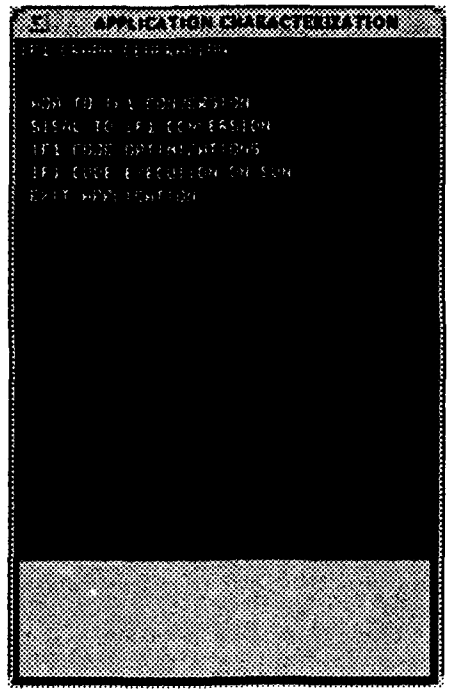


FIGURE 5 AIC Main Menu

4.1 Ada to IF1 Conversion

This option is selected by moving the mouse pointer over "ADA TO IF1 CONVERSION" in the application characterization menu and clicking the left mouse button. A menu of four options is displayed (see figure 6). The first option "PARSE ADA SOURCE PROGRAM ONLY" parses Ada programs without generating IF1 code. It is very useful in detecting correct Ada programs before any conversion. The sec-

ond option "CONVERT ADA TO IF1" is the core of the tool and its selection enables the user to convert Ada source code into IF1 code. The third option "VIEW ADA CONVERSION FILES ONLY" allows the user to view the files generated by a conversion. The selection of any of the options displays a menu of Ada files. There are three files generated after converting an Ada programs. These files store information about the application code as well as the conversion process.

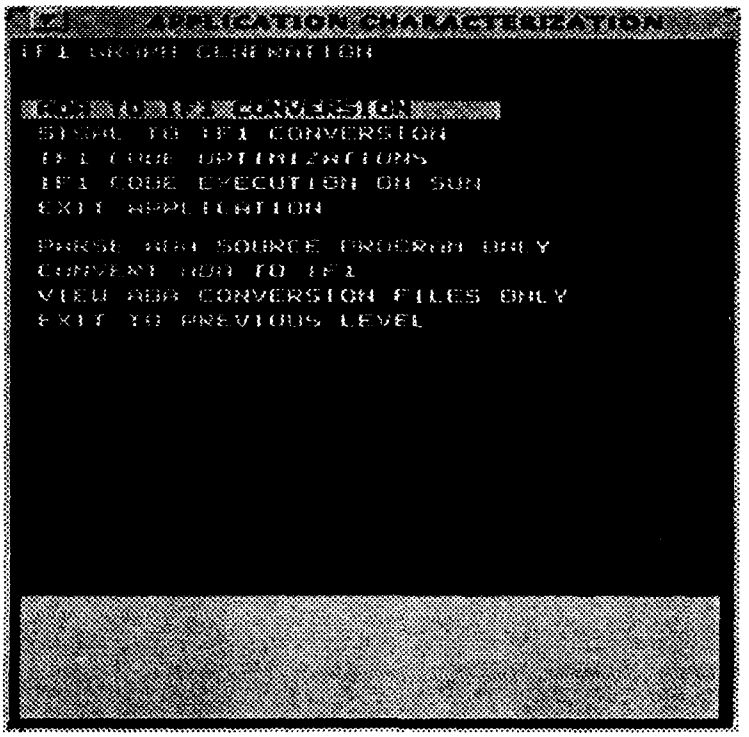


FIGURE 6 ADA to IF1 Conversion Options

4.2 Sisal to IF1 Conversion

This option is selected in the same manner as the "ADA TO IF1 CONVERSION" option. Sisal is an experimental applicative language similar to IF1. Applications written in Sisal can be transformed into parallel IF1 graphs. The conversion is done by using the OSC tool. This option displays a menu of Sisal files.

4.3 IF1 Code Optimizations

This option is selected in the same manner as the "ADA TO IF1 CONVERSION" option. This option is useful in certain cases to obtain equivalent IF1 codes. When this option is selected the user is presented eight conventional optimizations which can be applied to a selection of IF1 source files (refer to figure 7). One option enables the user to apply all the optimizations. Any selection made in this menu will display a menu which enables the user to view both the source and the target files.

4.4 IF1 Code Execution on Sun

This option is selected in the same manner as the "ADA TO IF1 CONVERSION" option. This option uses the OSC tool to run IF1 code on a Sun machine. Selection of this option displays the second menu list in figure 8. IF1 can be converted into three intermediate forms which are merely dataflow optimizations of IF1 for efficient execution. These are obtained by selecting the three first options. The fourth and the fifth options respectively, convert IF1 code to an equivalent C program and local Assembly code. The last option executes IF1 code on a local Sun machine. Both the source and target files can be viewed by using the menu.

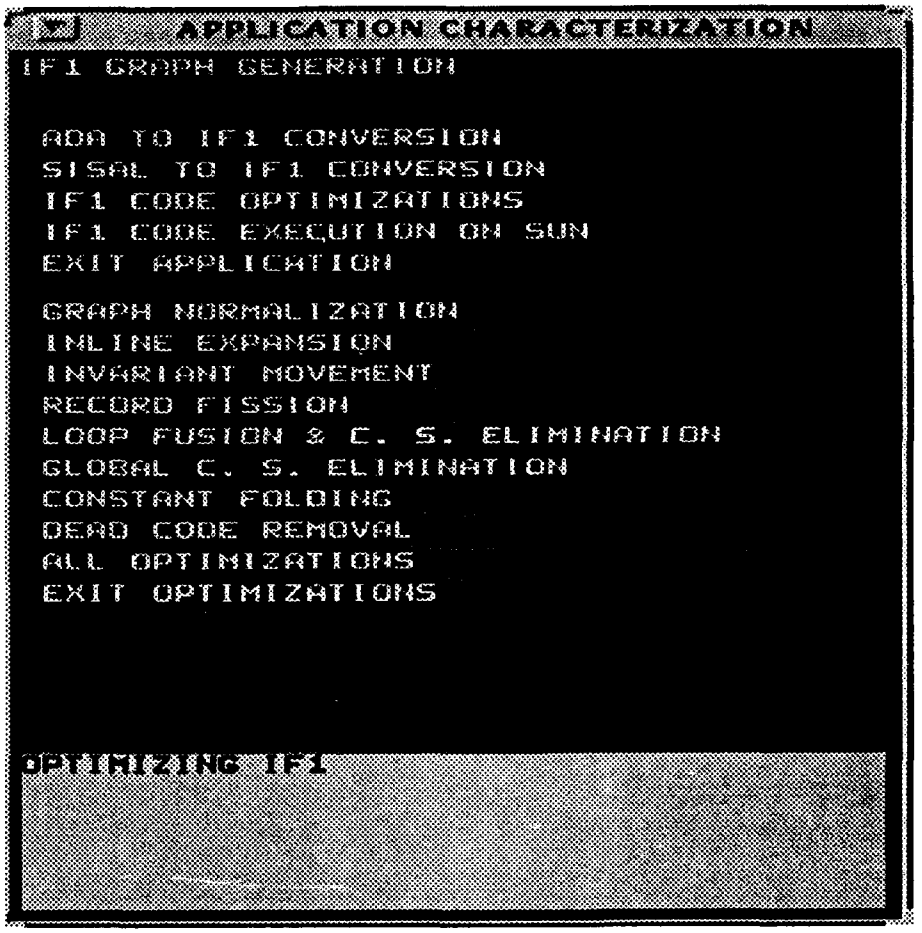


FIGURE 7 IF1 Optimization Menu

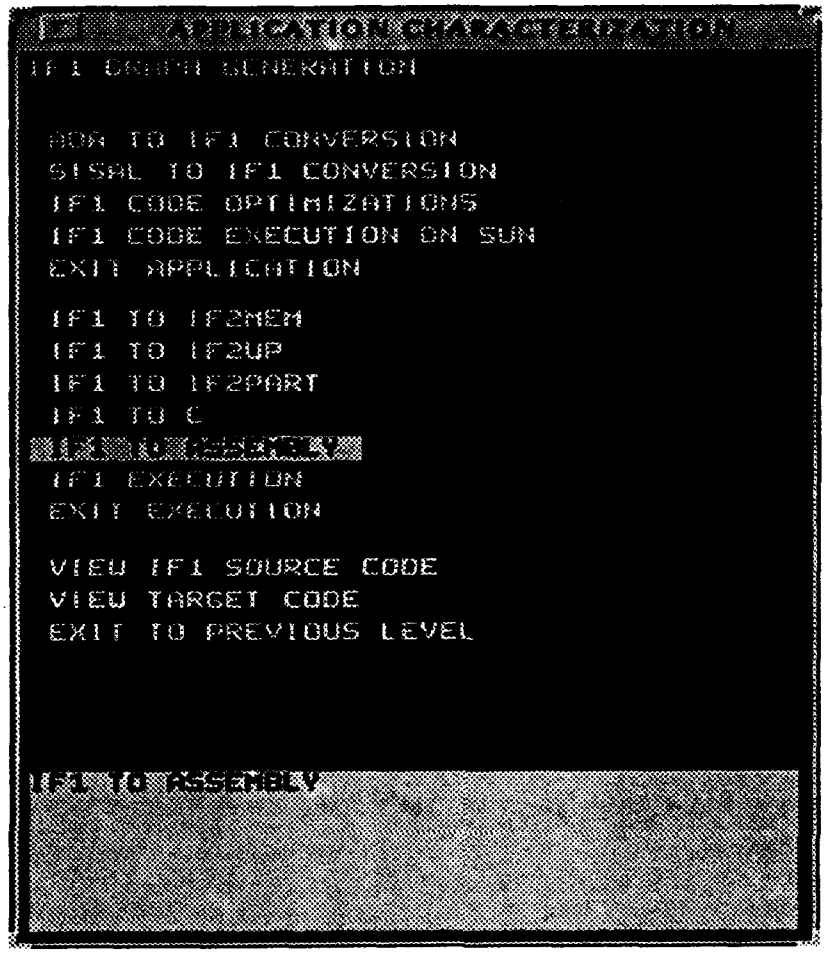


FIGURE 8 IF1 Code Execution On Sparc1+ Menu

5 Running The Graphics Display Tool (GDT):

The Graphics Display Tool (GDT) is started by moving the mouse pointer over "GDT (Display)" in the tools menu and clicking the left mouse button. The GDT menu will be displayed , (refer to figure 9).

GDT allows the user to display IF1 graphs , partition IF1 graphs and execute IF1 graphs.

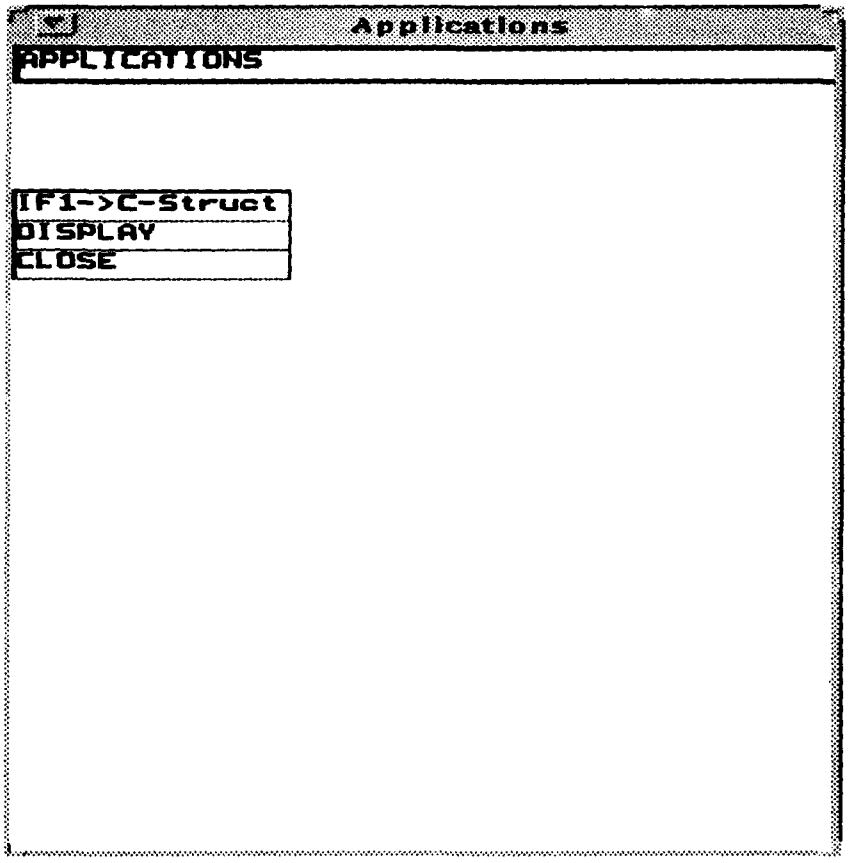


FIGURE 9 Graphical Display Toll Menu (GDT)

5.1 IF1->C_Struct:

This option is selected by moving the mouse onto the "IF1->C-Struct" in the GDT menu and clicking the left mouse button. When the "IF1->C-Struct" is selected, a file listing is displayed (refer to figure 10) which allows the user to select an IF1 file to be converted to a C-Struct.

5.2 Display

This option is selected in the same manner as the "IF1->C-Struct" option.

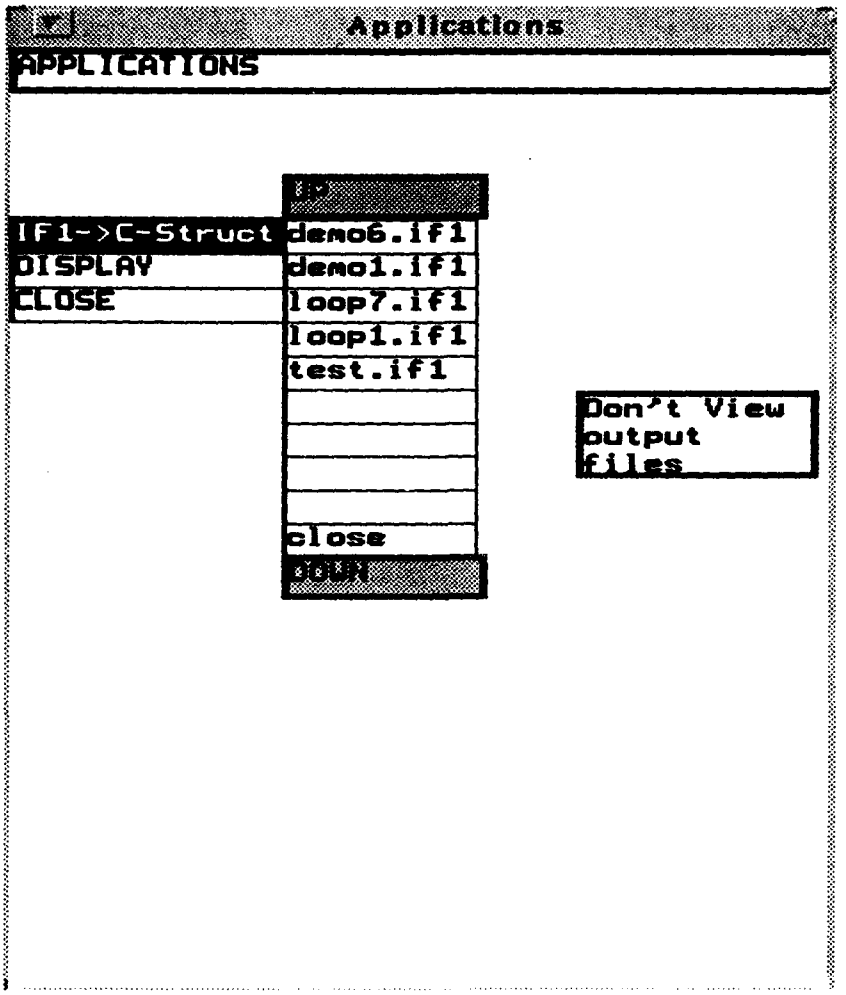


FIGURE 10 File listing in the GDT.

When "DISPLAY" is selected a file listing is shown. The user then selects a file for display. In the "DISPLAY" mode the user can manipulate the graph further through the GMM (Graph Manipulation Menu) which is displayed by pressing the middle mouse button. In figure11 the graph created by the GDT is given for the IF1 program While the cursor is in the display window, the user can invoke the GMM menu by clicking the middle mouse button and selecting a particular option.

The usage of each option will be detailed in section 5.2.1.

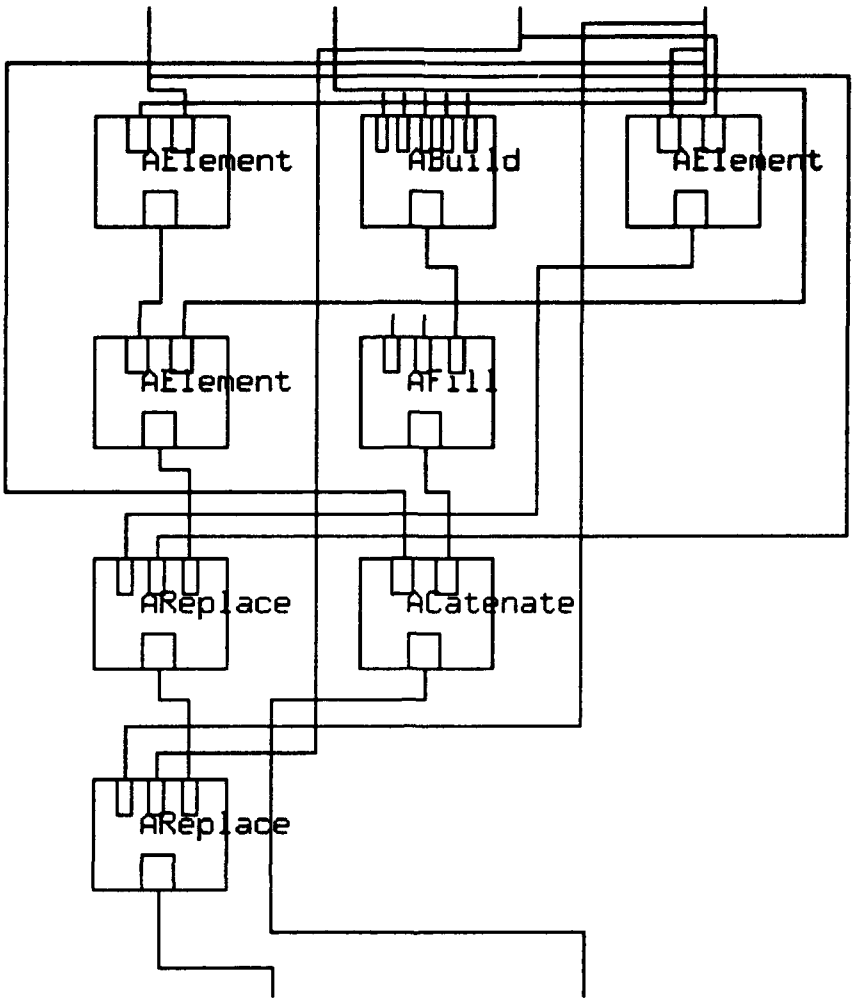


FIGURE 11 Displaying a Graph

5.2.1 Selecting and Unselecting a node, edge or literal

The user can select a node, edge or literal by clicking the left mouse button with the mouse pointer inside the node or the box which represents the edge or literal. When selected, a node, edge or literal changes color. To unselect a node , edge, or literal , the user can click the left mouse button in a clear area of the display window. A new node, edge or literal can then be selected by clicking the left mouse button with the mouse pointer inside the new node or box.

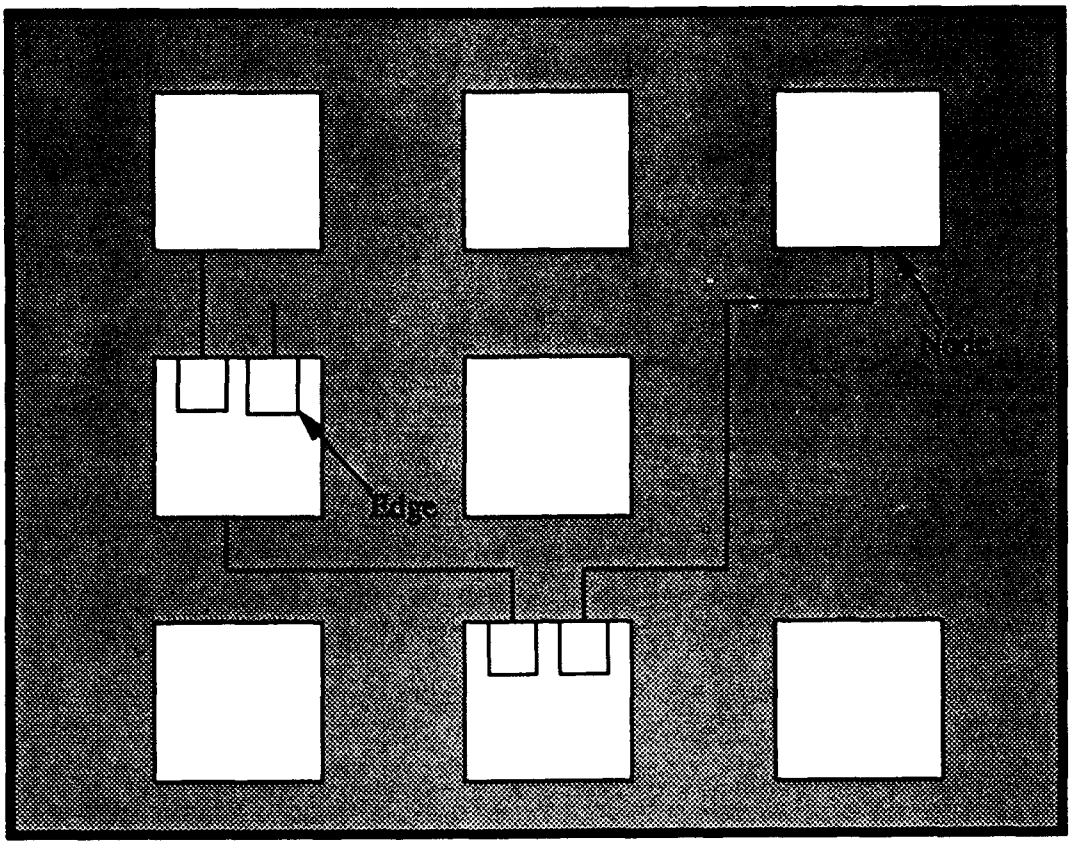


FIGURE 12 Node and Edge Graphical Representation of an IF1 Graph

5.2.2 Graph Main Menu (GMM)

GMM is invoked by clicking the middle mouse button while the mouse pointer is in the display window, refer to figure 13.

The GMM options provide complete access to the graph. They enable the user to examine any part of the graph in a hierarchical fashion.A description of the function of each of the GMM options follows:

- Expand: Displays the subgraphs of a compound node. A user must select a compound node before selecting the expand option.By expanding a compound node the IGD displays the scope immediately lower than the currently displayed scope.It also displays the current scope which can no longer be manipulated.

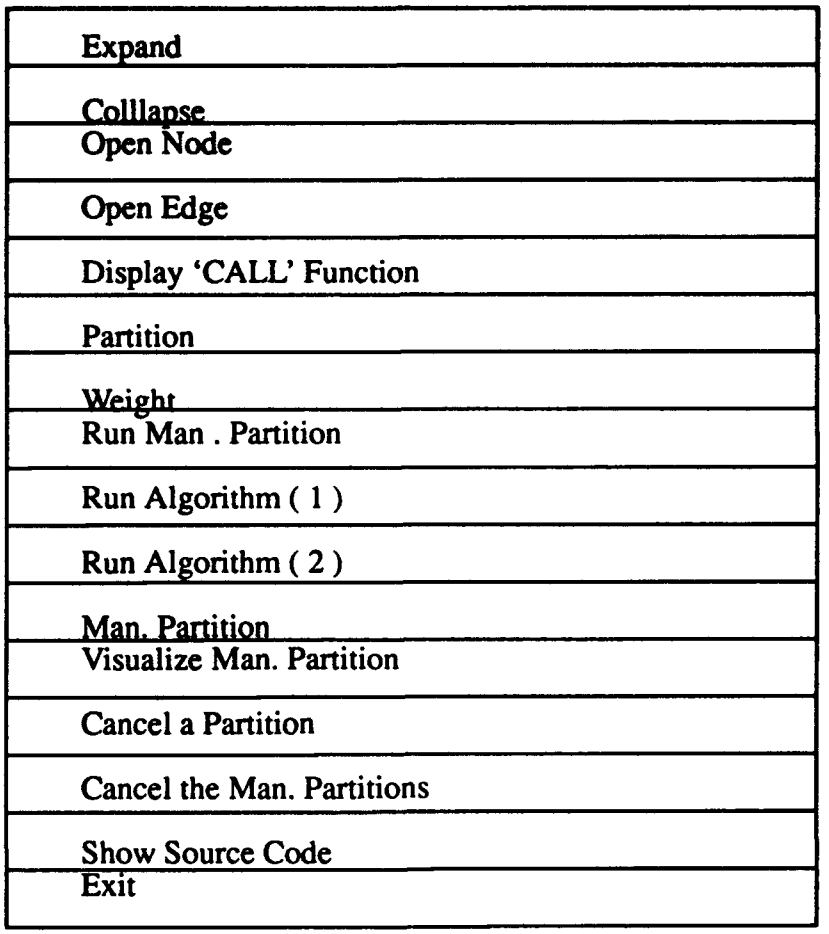


FIGURE 13 Graph Manipulation Menu (GMM)

- Collapse: Destroys the current scope and display the previous scope.
- Open node: Opens a node and displays information about the selected node . An example of information displayed includes: op_code, type of the node if it is compound or simple . The user must select a node first.
- Open edge: Similar to the open node option but it is for an edge.
- Display call func:Displays the graph of the function which is called by the call node selected. The user must select a call node in order to execute this option.
- Man Partition: Partitions the graph according to the user's selections.The user must choose a cut set of the nodes first by selecting an edge.The partitioning is done according to a partitioning algorithm which in brief adds a new parti-

tion to the graph each time "man partition" is executed succesfully.This new-man partition". (refer to figure 14).

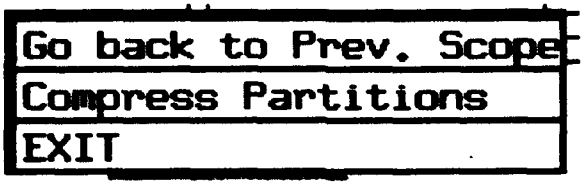


FIGURE 14 Intermediate Menu (IM)

If the "go to prev. scope" option is selected, the previous scope will be displayed. If the "compress partition" option is selected the nodes belonging to the same partition will be displayed in one box .If the user presses the middle mouse button with the mouse pointer in this display window, a new menu called the Partition Menu (PM)will be displayed, (refer to figure 15).

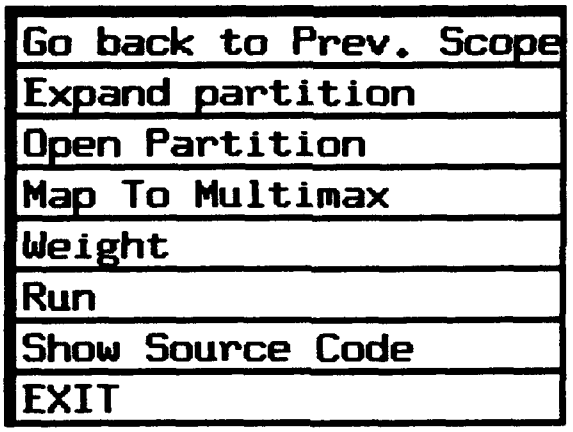


FIGURE 15 Partition Menu(PM)

The "go to prev. scope" option displays the IF1 graph at the latest scope viewed. The "expand partition" option will display the nodes of the selected partition (currently not in use). The "open partition" option will display information about the selected partition.The "map to Multimax" option will map the current partition of the graph to the Encore Multimax multiprocessor machine (currently not in use). The "show source code" option will display the source code of the selected partition(i.e. the source code of the nodes and edges that belong to that partition).The "quit" option takes the user back to the "Display" menu of the GDT.

5.3 The Error Messages

A listing of the error messages issued by the GDT is found in the appendix.

approach of algorithm (1) in section 5.5.1.

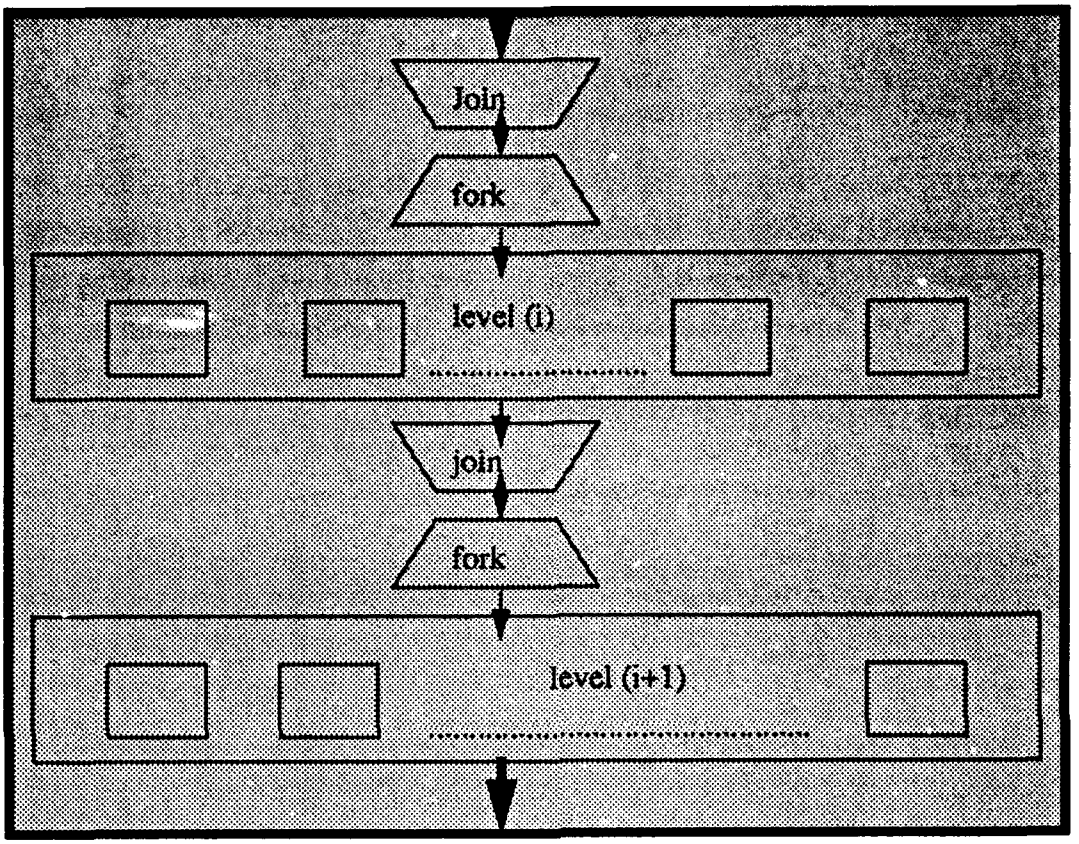


FIGURE 16 Join Fork Execution Fashion of the nodes in Levels

For both algorithm (1) and (2) the running of the program starts from the top level and continues until the completion of the bottom level. The user can interrupt the execution of the program by pressing any mouse button inside the window wherein the graph is displayed

5.4 Manual Partitioning of the IF1 Graph Code Display

5.4.1 Manual Partitioning of the IF1 Graph Code Display

In order to execute the IF1 graph on a given parallel system it should be partitioned into parallel tasks, then these tasks should be mapped onto the PE's of the parallel system. Usually partitioning is done automatically by the software.It is difficult to verify whether or not this automatic partitioning is efficient. One method of verify-

ing the performance of the automatic partitioner is to provide a tool that allows the user to manually partition the graph, then comparison can be made between the automatic and manual partitions.If the user partitioned a graph and obtained a better performance than the automatic partitioner then we can say that the automatic partitioner is not optimal and should be modified.
In this section we describe this type of a tool called the IF1 Manual Partitioner (IMP). In subsection 5.6.2 we present a general description of this IMP.

5.4.2 IMP General Description

As a starting point all nodes in the graph are assigned the zero partition. The user partitions the graph by selecting an edge that makes a cutset between the nodes.The source node of the selected edge is called the tnode which is the first node to be included in the new partition. The old partition of tnode is considered the parent or father of the new partition . The nodes included in the new partition include all nodes in the parent partition that have a path from their output node to the input of the tnode .The parent partition is then modified to exclude all the nodes assigned to the new partition.This approach insures that the partitioned tasks have no cycles among them since we start with a data flow graph which contains no cycles and enforce the no cycles policy during the partitioning. Figure 17 shows an example of the implementation of the IMP to a sample IF1 graph. There are two trivial partitions of the IF1 graph;

- All nodes belong to the same partition, then the number of partitions is equal to one.
- Every node belongs to a different partition, then the total number of partitions is equal to the total number of nodes in the graph.

An existing partition can be cancelled by using the option in the menu which manipulates the partitioning process. Also all the existing partitions can be cancelled at once and the user can start from the begining all over again.

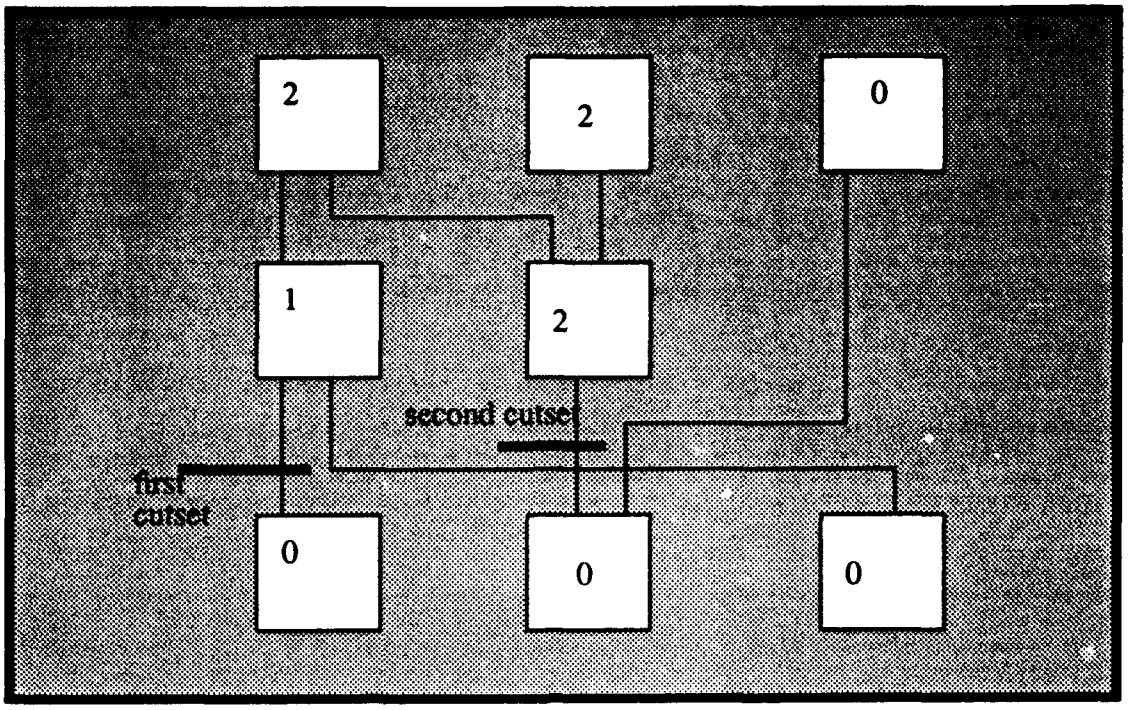


FIGURE 17 Example of Manual Partitioning of an IF1 Graph

6 USING the Predictive Parallel Assessment Tool (PPA)

PPA is an important part of PAWS. It allows the user to display parallelism profiles, speedup curves and metrics. It takes as input an IF1 file that was partially converted to a C structure. The PPA tool can be started by moving the mouse pointer over "PPA (Assessment)" from the tool menu and clicking the left mouse button. The predictive assessment tool window will be displayed (see figure 18)

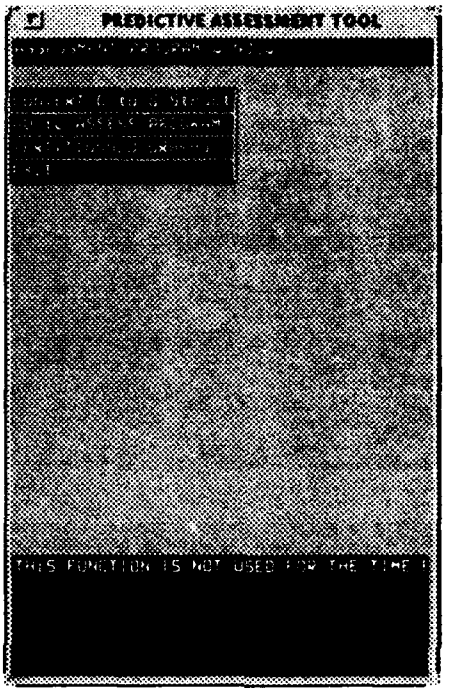


FIGURE 18 PPA Startup Window

6.1 CONVERT C TO G Struct

The assessment process starts by converting a C-Struct to a G-Struct. This task is accomplished by selecting the "CONVERT C to G-Struct" option.This option can be selected by moving the mouse pointer over the "CONVERT C to G Struct" and clicking the left mouse button.A listing of available files is displayed as showed in figure 19. The user can select a file for conversion. A message window at the bottom of figure 19 will display the current state of the PPA Tool. If more than ten files are available, then use the "UP" and "Down" button to scroll through the available files. All files must be IF1 files and must have the extension.if1. Once the conversion has been completed, the message window will display the word "READY".

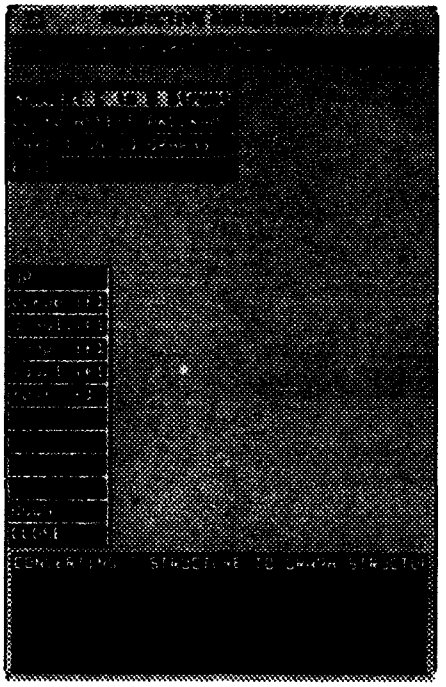


FIGURE 19 Choosing a File for Conversion

After a successful conversion, the user can invoke the "GO to ASSESS PROGRAM" option. This option is selected in the same manner as the "CONVERT C to G Struct" option. When the "GO to ASSESS PROGRAM" option is selected, a new menu is displayed over the current menu (see figure 20).

6.1.1 RUN ASSESS CODE

The "RUN ASSESS CODE" option is selected by moving the mouse pointer over it and clicking the left mouse button. When this option is selected a list of files is displayed and the user can select the appropriate file. Once the user selects a file and the run is successful, a "READY" message appears in the message window.

6.1.2 DISPLAY RESULTS

The option "DISPLAY RESULTS" is selected in the same manner as the "RUN ASSESS CODE" option.By selecting "DISPLAY RESULTS" a menu is displayed. All the options available at this stage are for display purposes and analysis, (see figure 21 shows the options available for result display.

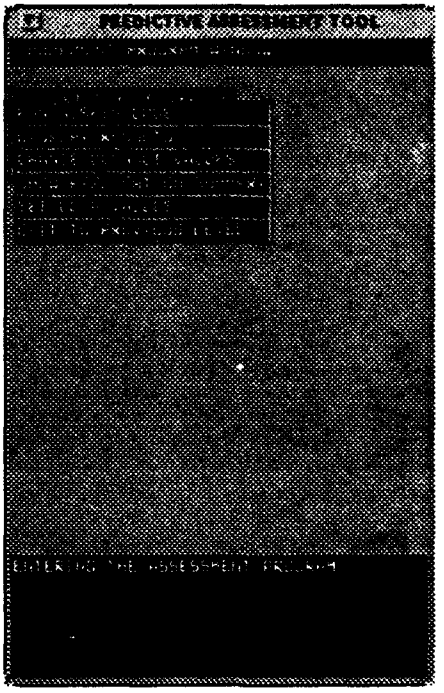


FIGURE 20 Go to Assess Program Options

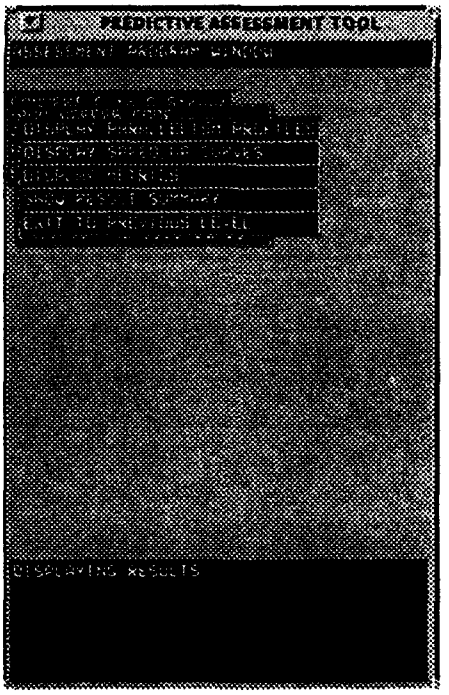


FIGURE 21 Display Result Options

6.1.3 DISPLAY PARALLELSIM PROFILE

By selecting this option, the user can display the parallelism profile of any compound node or call node that exist in the application. All files eligible for display have the following syntax <**profile**_name.**main_compound**.#> where the boldface components are reserved words of the PPA tool.The word "name" would be the name of the program being assessed and the number "#"would be the id number of the compound nodes that exist within the main program. All files that have some other word in place of main are call node display files.

6.1.4 DISPLAY SPEED_UP CURVES

By selecting this option, the user can display a cumulative speed_up curve for any compound node or call node that exist in the application. All files eligible for display have the following syntax <speed_up_name.**main_compound**.#> where the boldface components are reserved words of the PPA tool.The word "name" would be the name of the program being assessed and the number "#"would be the id number of the compound nodes that exist within the main program. All files that have some other word in place of main are call node display files.

6.1.5 DISPLAY METRICS

By selecting this option, the user can display the execution time of any main program or function contained in the main program.

6.1.6 SHOW APPLICATION SUMMARY

Displays in a condensed fashion the main parameters of the application the user is evaluating. To exit the user must type "close" at the bottom of the window, (see figure 21).

6.1.7 CHANGE DEFAULT VALUES

By invoking this option the user can change the database used by the PPA tool to evaluate an application. Currently three different databases exists. A user's database, in which data is custom fit to the user's needs, and two predefined databases for MIMD type machines (such as the Multimax), and SIMD type machines (such as the Connection machine). An on-line architecture tool has been designed to access all aspect of the architecture being evaluated.
Important: PPA uses by default the user's database. If a new database is chosen, then the program must be rerun before any effect can take place.

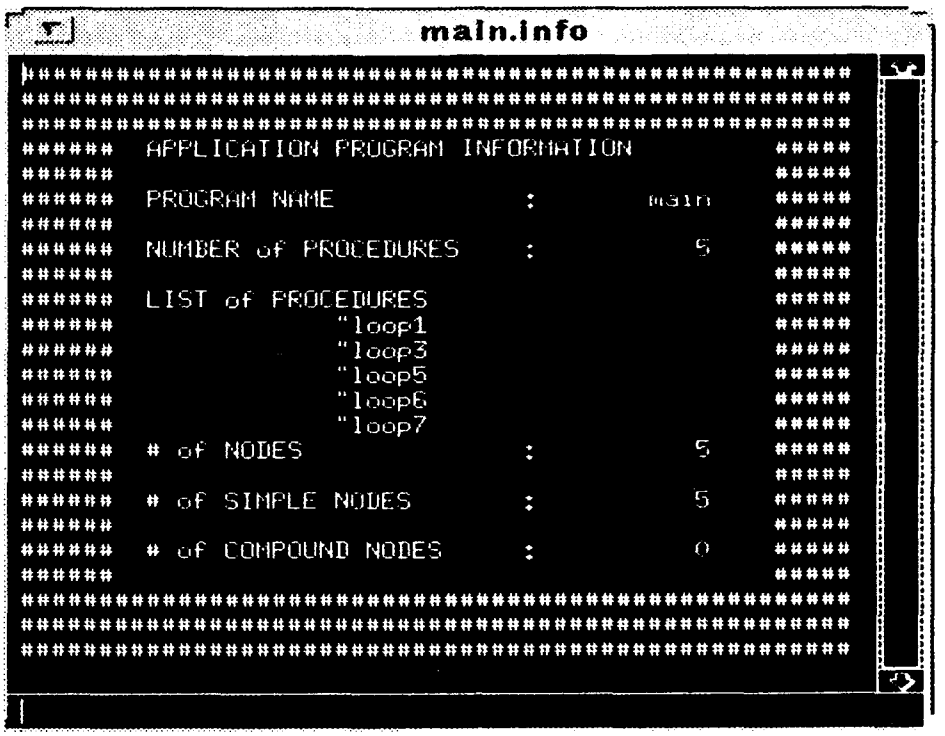


FIGURE 22 Result Summary Example

6.1.8 SET LOOP VALUES

PPA does not handle run time loop bounds. All loop values must be entered before running the assess program. A default value of five iterations is taken if bounds are not set by the user. If bounds cannot be determined at compile time, then an estimate or a determination from a frequency count should be applied. To select this option the user must move the mouse pointer over "SET LOOP VALUES" and click the left mouse button. A window similar to the window in figure 23 will be displayed.

In the example shown in figure 23, file loop6.2 contains three fields.
The first field is the type of LOOP (A or B). The second field is the upper bound for that node. The third field is an advanced feature allowing the user to specify if the loop can be sliced or not. The default value is False. In case, slicing is needed then that field must be replaced by the integer value "1".

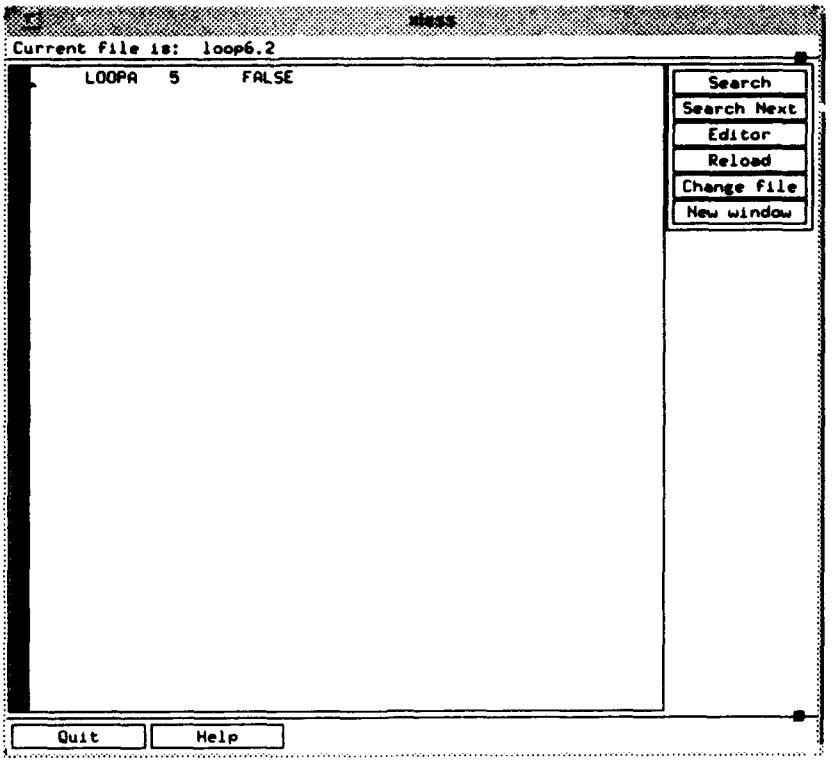


FIGURE 23 *Setting Loop Values*

In the example shown in figure 23, file loop6.2 contains three fields.
The first field is the type of LOOP (A or B). The second field is the upper bound for that node. The third field is an advanced feature allowing the user to specify if the loop can be sliced or not. The default value is False. In case, slicing is needed then that field must be replaced by the integer value "1".

APPENDIX

-A- Supplemental User's Manual for PPA

A.1 File Format:

This part will be explained through the use of an example. Assume an Ada program called **"application.a"** exists. After running it through the ADA to IF1 converter, it generates a file called **"application.if1**.Assume that **"application.if1"** contains a main program and two procedures called **"procedure1"** and **"procedure2"**. Assume that **"procedure1"** contains two compound nodes. Conversion of the IF1 structure to the G structure will generate the following files:

- "application.data",
- "application.main",
- "application.procedure1",
- "application.procedure2"

The file **"application.data"** will look like figure 24
If the number of nodes in procedure1, procedure2 and **"application"** were 34,16 and 6 respectively then the file "application.dat" will contain the following 4 lines (see figure 24):

3	
Appplication.procedure1	34
Application.procedure2	16
Application.main	6

FIGURE 24 Example of Application Data

A.2 What is the content of the 3 generated files?

Each one of the three cited files contains data directly relevant to the application at hand. As a simple reminder we would like to emphasize the fact that nodes are the focal point, and therefore each node will be associated with a set of data that describes its location, number of inputs and outputs, execution time, and level. Assuming the G structure has been properly generated, then the assessment program, can be invoked. A successful run of the assessment program will generate several files that will be used for plotting items such as parallelism profiles, and speed_up curves. For the example at hand, this would generate the following files:

- profile_application.main
- profile_application.procedure1
- profile_application.procedure1_compound.#1
- profile_application.procedure1_compound.node#2
- profile_application.procedure2

These files will be stored in the directory $MYPAWS/PPA/PROFILES/PARAL-PROFILES. The speed_up data files will be stored in the directory $MYPAW/PPA/PROFILES/SPEED-UP. The files expected to be found are:

- speed_up_application.main,
- speed_up_application.procedure1
- speed_up_application.procedure1_compound.#1
- speed_up_application.procedure1_compound.#2
- speed_up_application.procedure2

All of these files are displayed using the **xgraph** utility of the X Window system. The directory $MYPAWS/PPA/PROFILES/EXECUTION-TIMES will contain the following files:

- application.main,
- application.procedure1,
- application.procedure2

These files contain the execution time of the 2 procedures as well as the execution time of the main application.

A.3 A Simple Example: Matrix Multiplication

The code of matrix multiplication is shown in figure 25.

```
for(i=0;i<N;++i)
    for(j=0;i<N;++j)
    {
    sum=0;
    for(k=0;k<N;++k)
            M[i][j]=sum + A[i][k]*B[k][j];
    }
```

FIGURE 25 Matrix Multiplication Example

For obvious reasons,the inner most loop cannot be executed in parallel as written. However, if the user specifies a loop slicing procedure (See section 6.1.8), then erroneous results will be generated without the user detecting any anomaly in the execution.

Figure 26 and 27 show the parallelism profile of the matrix multiplication program without loop slicing and with loop slicing.

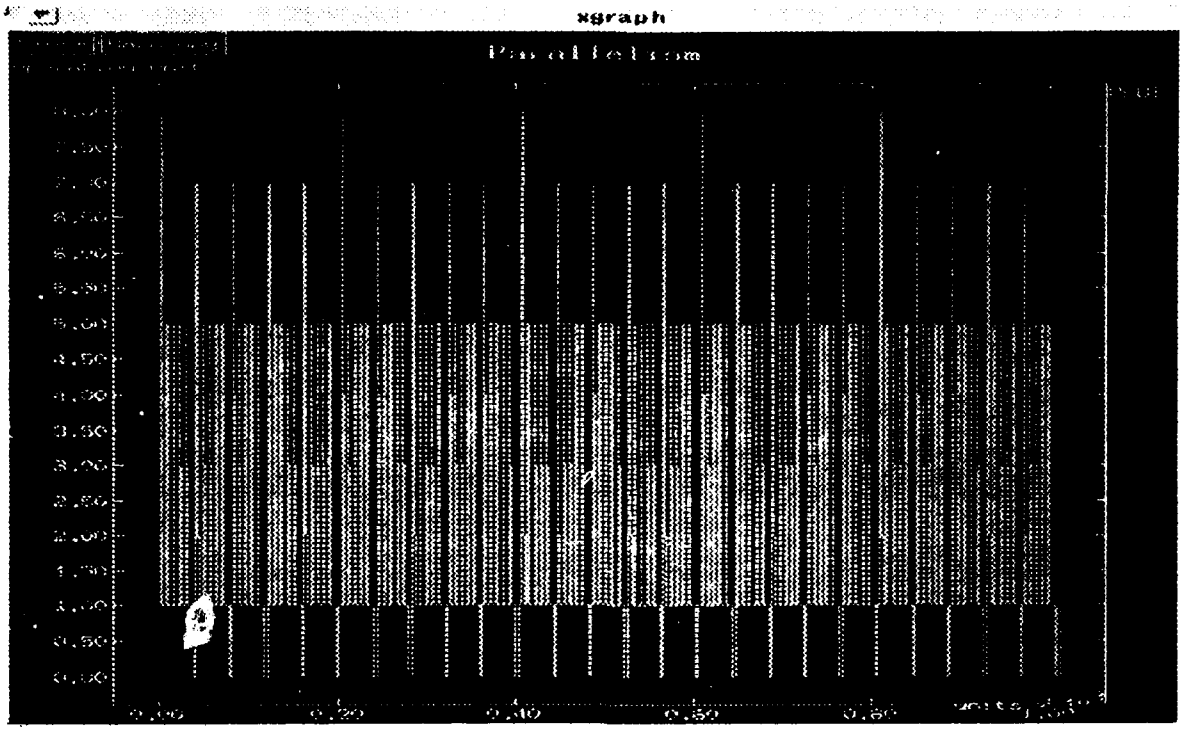


FIGURE 26 Matrix Multiplication without Loop Slicing

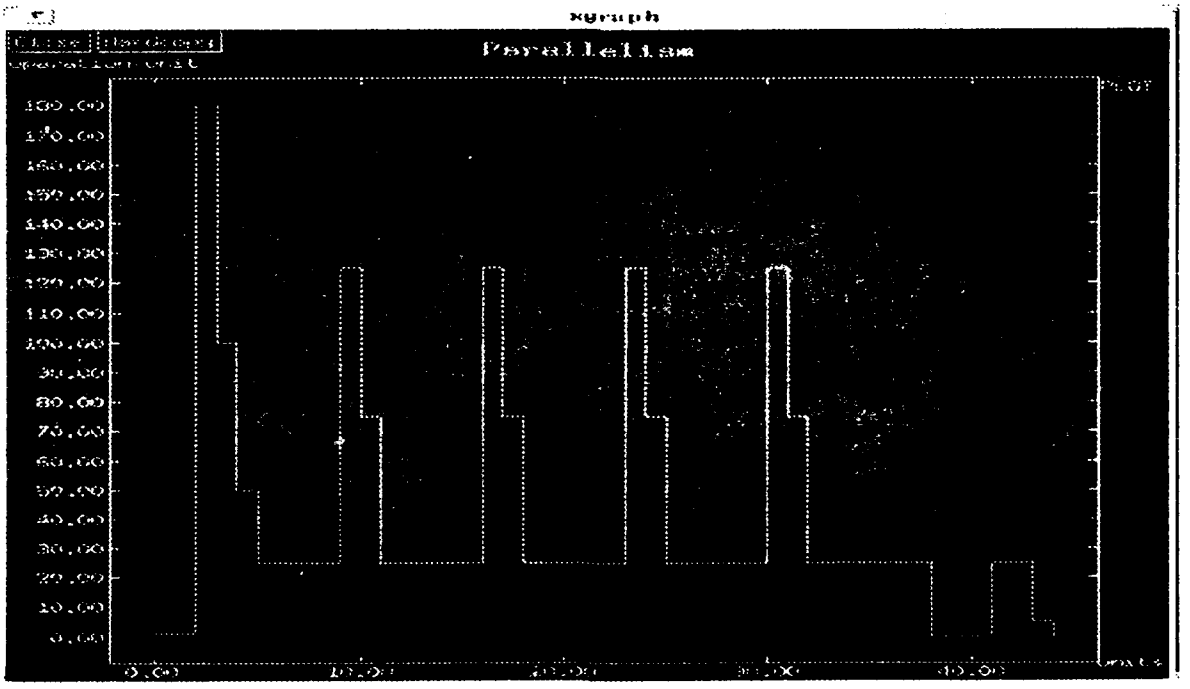


FIGURE 27 Matrix Multiplication with Loop Slicing

-B- IGDT APPENDIX

B.1 THE IF1 to C-STRUCTURE PARSER (ICP)

An IF1 to C-structure Parser (ICP) tool has been designed which is described in this section. This ICP takes an IF1 file as an input and produces a complete set of C-structures as an output. This completely describes and represents the IF1 code. In this section the ICP tool is described. The ICP scans the IF1 program file and constructs a data structure representing the program as a result. Two files containing information about the constructed data structures are generated. The ICP scans the IF1 file and builds a list of nodes, edges and graphs. The list of the graphs represent the main graph in the IF1 file and the function graphs corresponding to the call functions.The data dependency between the nodes is empirically included in each node structure.The list of edges includes all the edges and literals in the IF1 file.

B.2 C Structure Definitions

These structures are defined to contain the information needed to create the visual data flow graph and to parallelize it; specifically there are three fundamental structures:

- Node structure
- Edge structure
- Graph structure

B.3 Accessing Information In The Data Structures

Three lists of data structures are built: one for the nodes "nodeptr[][], one for the edges "edgeptr[][]" , and the third is for the graphs "graphptr[]". The first dimension in the nodeptr and edgeptr lists is the scope specification and the second dimension is the node or edge index within the scope.There are two ways to access the information in the IF1 graph of the data structures returned by the ICP:

- The first way to access the information is where the data structures are accessed directly. For example if the user wants to access element x in the node j in scope i the he is to print nodeptr[i][j].x.
- The second way to access the information is an indirect one where the user is to use the displayed graph to get the information. For example if the user wants to access the information about a specific node in the graph the user must select that node and open it to get the information.

B.4 IF1 Graph Display(IGD) Tool Approach

The IGD consists of two sub-tools:

- The IF1 to C structure parser (ICSP) sub-tool.
- The C structure to IF1 graph display (CSIGD) sub-tool

The IGD gets the IF1 code as input and generates the IF1 graph as output. This is done in two steps (see figure 28):

- ICSP takes the IF1 code as input and generates a C structure as output.
- CSIGD takes the generated C structure as input and generates the IF1 graph as output using the graph manipulation menu (GMM)..

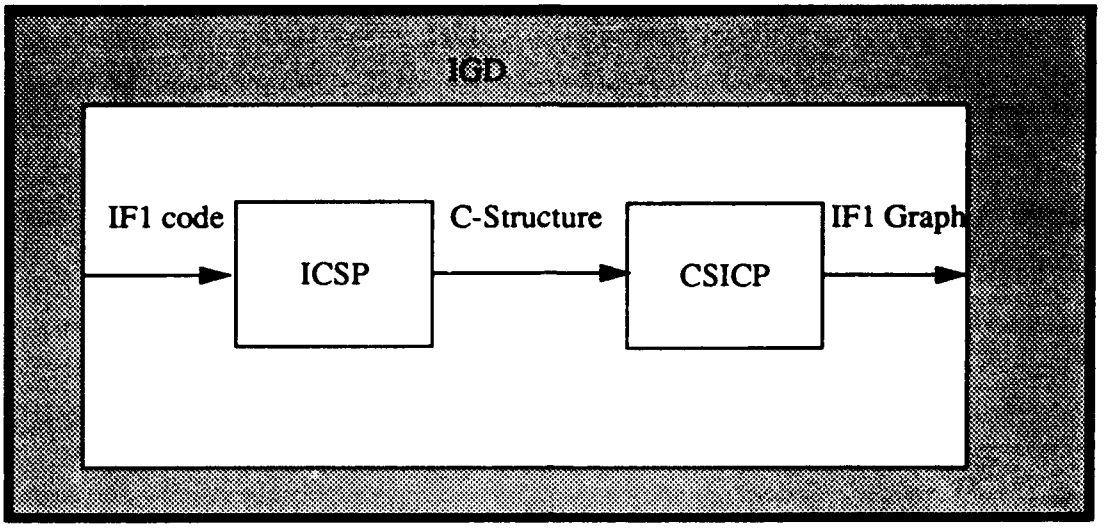


FIGURE 28 IF1 Graph Display (IGD) Tool. General View

The user selects a file.When doing so the IF1 graph corresponding to the IF1 file chosen is displayed. To further manipulate the graph the user must access the GMM by pressing the middle mouse button inside the window containing the graph. The IF1 graph will be displayed hierarchically i.e. at different scopes of complexity. The highest scope (scope 0) represents the less detailed one and the lowest scope (scope n) represents the most detailed one.The position of every node in the graph is determined uniquely by its level and position in the graph. If there are four nodes N1,N2,N3 and N4 in level i for example then they are given positions 0,1,2,and 3 respectively in that level. All the nodes at the same level will be displayed at the same horizontal location so that the user can visualize that these nodes can be executed in parallel (i.e. there is no data or control dependency between them). If a node Ni should be executed before node Nj then Nj depends on node Ni and Ni should be put in a higher level, where level0 is the highest level and level(n-1) is the lowest level if the total number of levels in the graph is n.A higher level of nodes is displayed in a higher horizontal location. Figure 29 shows what the IF1 graph will look .In Figure 29 there are K +2 horizontal channels where K +1 is the total number of levels in the graph;and p+2 vertical channels, where p +1 is the maximum number of nodes in any level. These channels are designed to manage displaying the IF1 graph edges, and literals. The total number of horizontal channels is equal to the total number of levels minus one, and the total number of vertical channels is equal to the maximum number of nodes in any level plus one. We will present later how these channels are used to manage displaying the edges and literals.The IGD tool

receives the IF1 code as C structures and produce the IF1 graph as an output as shown in Figure 29.The nodes are positioned in the window according to their level and level position. To position the nodes their x and y coordinates as well as their width and height are specified. All the nodes are of the same size. The width and height of the nodes depend on the size of the window, the total number of nodes in the graph, the total number of levels, and the maximum number of nodes in all the levels

The edges are positioned according to their source and destination nodes as well as the source and destination ports. The edges consist of line segments. If the destination node of some edge is in the next level of the source node of that edge then the total number of line segments of that edge is three otherwise it is five as shown in Figure 29. Line segments of the edges do not cross the nodes but pass through the horizontal and vertical channels. To position the edges we must first build channels for each graph, then the positions.

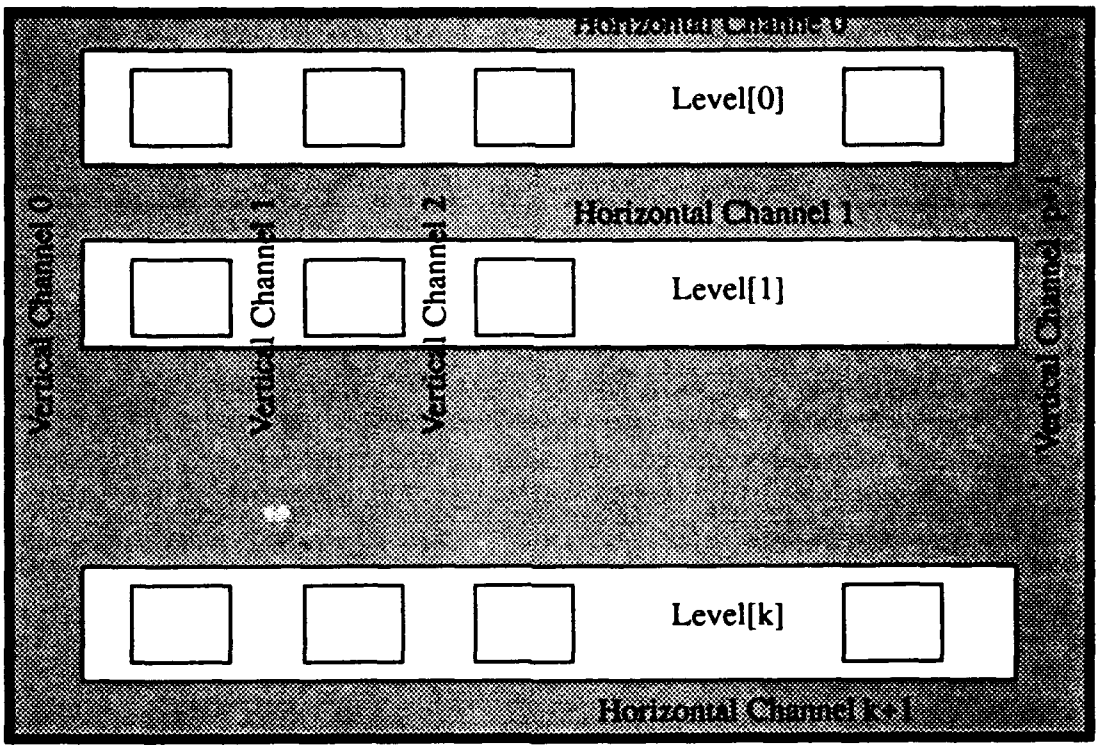


FIGURE 29 General View of the Displayed IF1 Graph

After positioning the nodes and edges, and while the window where these nodes and edges are to be drawn is created and mapped, we must use the X window system functions to draw these nodes and edges in the window. We also draw the node operation codes inside these nodes so that the user can understand the graph properly

B.5 error messages

When the user tries to do something which is not allowed, a new window will pop up displaying an error message. This error message tells the user what should be done first in order to be able to accomplish the command sucessfully. An example of an error is when a user attempts to expand a simple node in an IF1 graph. If a user attempts this the following possible list of eror messages will be displayed:

- Can't expand; Last scope
- Can't collapse; Top scope.
- Please select a node first.
- Please select an edge first
- Please select a compound node first.
- Please select a call node first.
- Please select a partition first.
- The partition already exists. Choose another edge.
- Please select an edge with a REAL source node.
- Please select an edge and not a literal first.
- Please select a node or edge first.